**Создание класса: свойства и методы**

Рассмотрим пример создания простейшего класса. Давайте с его помощью смоделируем квадраты на координатной плоскости.

Каждый квадрат будем определять своим центром (т.е. точкой с двумя числовыми координатами) и стороной (т.е. её длиной, представляемой в виде числа). Таким образом, квадрат на координатной плоскости характеризуют 3 вещественных числа. Значит в нашем классе должно быть три соответствующих свойства.

Не будем пытаться решать серьёзных задач с помощью класса, а наделим его следующими возможностями: созданный на основе класса квадрат должно быть возможно выводить на экран (в виде описания его характеристик), перемещать (т.е. совершать преобразование движения, меняя координаты его центра) и масштабировать (т.е. совершать преобразование подобия, меняя сторону квадрата).

// описываем отдельный новый класс  
class Square {  
    // свойства класса  
    public double x; // абсцисса центра  
    public double y; // ордината центра  
    public double dl\_s; // сторона  
    // методы класса  
    // выводит на экран параметры квадрата  
    public void printSquare() {  
        System.out.println("Квадрат с центром ("+x+";"+y+") и стороной "+dl\_s);  
    }      
    // перемещает центр  
    public void moveSquare(double a, double b) {  
        x = x + a;  
        y = y + b;  
    }  
    // масштабируем, выполняем преобразование подобия с коэффициентом k  
    public void zoomSquare(double k) {  
        dl\_s = dl\_s \* k;  
    }      
}  
  
// описываем основной класс, содержащий метод main  
public class Main {  
    public static void main(String[] args) {  
        // Создаём объект (квадрат класса Square), у него будет нулевая  
        // сторона и центр в (0.0;0.0), поскольку все свойства получат  
        // значения по умолчанию  
        Square s1 = new Square();  
        // выводим на экран параметры квадрата  
        s1.printSquare();  
        // Меняем абсциссу центра, обращаясь к свойству x  
        s1.x = 3;  
        // Меняем сторону, обращааясь к свойству dl\_s  
        s1.dl\_s = 12.3;  
        // выводим на экран обновлённые параметры окружности  
        s1.printSquare();  
        // Создаём другой объект того же класса  
        Square s2 = new Square();  
        s2.dl\_s = 3.14;  
        s2.zoomSquare(1.66);  
        s2.printSquare(); // Квадрат с центром (0.0;0.0) и стороной 5.2124  
    }  
}

**Конструкторы**

Когда мы создаём объект командой Square s1 = new Square(); используется так называемый конструктор по умолчанию (или конструктор без параметров) — это специальный метод класса, мы его не определяли явно, но даже если его не определить он создаётся автоматически, выполняется при создании каждого нового объекта и присваивает первоначальные значения его свойствам (инициализирует их). Значения по умолчанию для свойств зависят от их типа (0 или 0.0 для числовых типов, false для логического типа и т.д.).

Конструктор по умолчанию можно описать явно и при этом задать начальные значения для свойств нового объекта, отличные от значений по умолчанию.

От остальных методов конструктор отличается тем, что имеет то же самое имя, что и весь класс, а также не имеет типа возвращаемого значения (по сути, в результате своей работы конструктор возвращает новый объект нужного класса).

class Square {  
    public double x; // абсцисса центра  
    public double y; // ордината центра  
    public double dl\_s; // сторона  
  
    public void printSquare() {  
        System.out.println("Квадрат с центром ("+x+";"+y+") и стороной "+dl\_s);  
    }      
    public void moveSquare(double a, double b) {  
        x = x + a;  
        y = y + b;  
    }  
    public void zoomSquare(double k) {  
        dl\_s = dl\_s \* k;  
    }  
    // конструктор по умолчанию, теперь сразу после создания объекта будем  
    // получать квадрат с единичной стороной с центром в начале координат  
    public Square() {  
        x = 0.0;  
        y = 0.0;  
        dl\_s = 1.0;  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
       Square s1 = new Square();  
        s1.printSquare(); // Квадрат с центром (0.0;0.0) и стороной 1.0  
    }  
}

Поскольку методы можно перегружать, а конструктор является методом, то с помощью перегрузки можно создать дополнительные варианты конструкторов. Например, удобно иметь конструктор, который позволит при создании объекта явно указывать координаты его центра и длину стороны.

Описать подобный конструктор можно в дополнение к основному следующим образом:

    public Square (double a, double b, double s) {  
        x = a;  
        y = b;  
        dl\_s = s;  
    }

Теперь при создании объектов можно пользоваться любым конструктором на выбор:

       Square s1 = new Square();  
        s1.printSquare(); // Квадрат с центром (0.0;0.0) и стороной 1.0  
       Square s2 = new Square(1,-1,14);  
        s2.printSquare(); // Квадрат с центром (1.0;-1.0) и стороной 14.0

Нужно учитывать следующий факт: если в классе описан явно хотя бы один конструктор с параметрами, то конструктор по умолчанию (без параметров) создаваться автоматические уже не будет (его в такой ситуации надо описывать явно). Хотя, если вам требуется только конструктор с параметрами (как второй из нашего примера), то можно обойтись и совсем без конструктора по умолчанию (описать в классе только один конструктор с параметрами).

**Доступ к членам класса из тела методов**

Добавим в наш класс метод, вычисляющий площадь того квадрата, к которому метод применён. Метод будет описывать так:

    public double squareSquare () {  
        double s = dl\_s \* dl\_s;  
        return s;  
    }

Результат работы метода можно увидеть следующим образом:

        System.out.println("Площадь квадрата s2: "+s2.squareSquare());

Обратите внимание: внутри каждого метода класса доступны свойства того объекта, для которого метод будет вызываться. То есть если мы вызываем метод для объекта s2, то внутри метода при его выполнении мы будем работать именно со свойствами объекта s2 (s2.x будет доступно x, s2.dl\_s будет доступно как dl\_s и т.д.).

Может возникнуть ситуация, когда для формальных параметров метода вы захотите использовать имена уже принадлежащие свойствам класса.

Например, можно было бы начать описание метода для масштабирования таким образом:

public void zoomSquare(double dl\_s) {…

Как же в таком случае обращаться к свойствам объекта (ведь имена этих свойств перекрываются формальным параметром)?

Решение такой неоднозначности существует: к любому свойству внутри метода можно обращаться не только по имени, но и через ссылку **this**. То есть внутри метода можно написать **x=13;**, а можно **this.x=13;** — эффект будет идентичный. Соответственно, когда имя формального параметра перекрывает имя свойства, к имени свойства нужно обращаться через ссылку **this**. Тогда метод можно переписать таким образом:

    public void zoomSquare(double dl\_s) {  
        this.dl\_s = this.dl\_s \* dl\_s;  
    }

Понятно, что удобнее не допускать перекрывания имён свойств именами локальных параметров в методах. Иногда, впрочем, требуется внутри метода применить какой-то другой метод к текущему объекту, тогда без ссылки this не обойтись.

Добавим в класс метод, проверяющий, совпадают ли два квадрата по площади.

В этом методе должны участвовать два объекта: тот, для которого метод вызван и второй участник сравнения, который может быть передан в метод через параметр. При этом параметр будет иметь соответствующий тип (не какой-то встроенный, а в виде класса Square).

Метод можно описать так:

    public boolean equalsSquare(Square squ) {  
        if(this.squareSquare() == squ.squareSquare()) {  
            return true;  
        } else {  
            return false;  
        }  
    }

Пример использования метода:

        if(s1.equalsSquare(s2)) {  
            System.out.println("Квадраты s2 и s1 имеют равную площадь");  
        } else {  
            System.out.println("Квадраты s2 и s1 имеют различную площадь");  
        }

**Задачи**

1. Создайте в классе Square метод, вычисляющий периметр.
2. Создайте в классе Square метод, перемещающий центр квадрата в случайную точку квадрата координатной плоскости с диагональю от [-99;-99] до [99;99]. Обратите внимание на то, что требуется создать обычный метод, применимый к уже существующему объекту, а не конструктор, создающий новый объект. Вызов метода **Math**.random() возвращает псевдослучайное вещественное число (**double**) из диапазона [0;1), то есть, от 0 до 1 исключая 1, а значит максимальное число в диапазоне это 0.99999999999...
3. Создайте в классе Square метод, вычисляющий расстояние между центрами двух квадратов.

Созданные методы протестировать в классе Main.

**Пример**

class Square {  
    public double x; // абсцисса центра  
    public double y; // ордината центра  
    public double dl\_s; // сторона  
  
    public void printSquare() {  
        System.out.println("Квадрат с центром ("+x+";"+y+") и стороной "+dl\_s);  
    }      
    public void moveSquare(double a, double b) {  
        x = x + a;  
        y = y + b;  
    }  
    public void zoomSquare(double dl\_s) {  
        this.dl\_s = this.dl\_s \* dl\_s;  
    }  
    public Square() {  
        x = 0.0;  
        y = 0.0;  
        dl\_s = 1.0;  
    }  
    public Square(double a, double b, double s) {  
        x = a;  
        y = b;  
        dl\_s = s;  
    }    
    // метод вычисляющий площадь квадрата  
    public double squareSquare() {  
        double s = dl\_s \* dl\_s;  
        return s;  
    }  
    // метод проверяющий равны ли квадраты по площадям  
    public boolean equalsSquare(Square squ) {  
        if(this.squareSquare() == squ.squareSquare()) {  
            return true;  
        } else {  
            return false;  
        }  
    }      
}  
  
public class Main {  
    public static void main(String[] args) {  
       Square s1 = new Square();  
        s1.print Square(); // Квадрат с центром (0.0;0.0) и стороной 1.0  
       Square s2 = new Square(1,-1,14);  
        s2.print Square(); // Квадрат с центром (1.0;-1.0) и стороной 14.0   
        System.out.println("Площадь квадрата s2: "+s2.squareSquare());  
        s1.zoomSquare(14);  
        if(s1.equalsSquare(s2)) {  
            System.out.println("Квадраты s2 и s1 имеют равную площадь");  
        } else {  
            System.out.println("Квадраты s2 и s1 имеют различную площадь");  
        }  
    }  
}